A Method and Apparatus for Wire-Speed Application Layer Classification of Data Packets

I. DESCRIPTION OF THE INVENTION

A. Field of the Invention

The present invention relates generally to the classification of packets in a full duplex communication system, and more specifically to high speed digital communication networks transporting packets which may be monitored at the application level of the communication model where wire speed handling of the packet is required. The present invention is embodied in a network system, a data classifier, a method for hashing and computer program products for enabling a computer to perform data packet classification.

B. Background of the Invention

In most communication networks used for exchanging messages between a source and a destination, a message in a digital form is divided into multiple packets for faster and more convenient transmission and for reducing errors. Examples of such communication networks are the Internet, Wide and Local Area Networks. The digital packets are then transmitted over the network between the source computer and a destination computer. It should be noted that the source and destination could be personal computers or servers and the like.

A modern day computer network system comprises a substantial number of individual computers and servers. A simple computer network is shown in FIG.1. Typically, a computer can act as both a source and a destination.

Computer sending a data packet is a source and a computer receiving a packet is a destination. The same computer that acted as a source in a packet transfer could act as a destination for another packet. A computer acting as a router acts as both the source and a destination for a packet. This is because it receives a packet from a different source and then simply routes it to a different computer.

Often, each of the computers in the network forms at least part of a "node" of the network, and data is transferred among the various nodes by transmitting data packets among the computers. For example, a first computer located at a first node may run a first application program that generates first data to be subsequently processed by a second computer at a second node. In order to transfer the first data to the second computer so that it can be processed, the first computer divides the first data into a plurality of data segments and forms a data packet corresponding to each of the data segments. Then, the data packets are transmitted downstream from the first computer to the second computer. Also, if the network is capable of full duplex communications, the second computer may transmit data packets upstream to the first computer in response to the data packets received from the first computer.

Each of the data packets transmitted from the first computer to the second computer (and transmitted from the second computer to the first computer) typically contains a data packet header. The headers are often referred to as tuples. The header often includes data that identifies the type of data contained in the data packet, the source computer from which the data packet was transmitted, the intended destination computer of the data packet, etc. An example of a data packet header is illustrated in Fig. 4.

As shown in the figure, the header HDR comprises a source internet protocol ("IP") address field 100, a destination IP address field 110, a protocol field 120, a source port field 130, and a destination port field 140. The source IP address field 100 contains a 32-bit source IP address that identifies the source computer transmitting the data packet. The destination IP address field 110 contains a 32-bit destination address that identifies the intended destination computer of the data packet. The protocol field 120 contains eight bits of protocol data that identify the data format and/or the transmission format of the data contained in the data packet. The source port field 130 includes sixteen bits of data that identify the computer port that physically outputs the data packet, and the destination port field 140 contains sixteen bits of data that represent the computer port that is supposed to input the data packet. The tuple uniquely defines the path between the source and destination and therefore defines the origin and target for the packet being sent.

When data packets are transmitted over the network from the source computer to the destination computer, they are input by various network components that process the data packets and direct them to the appropriate destination computer. Such network components may be included in the destination computer and/or may be contained in an intermediate computer that processes the data as it is being transmitted from the source computer to the destination computer. If the data packets can be quickly and efficiently processed and routed between the various nodes of the network, the operation of the entire network is enhanced. For example, by quickly and efficiently transmitting data packets to the destination computer, the quality of real-time applications such as internet video conferencing and internet voice conferencing is improved. Also, the network components can quickly process the data

packets to determine if they are authorized to be transmitted to the destination computer, and if they are not, the network components discard the data packets. As a result, the security of the network is greatly enhanced.

Before processing a data packet, a network component must "classify" the data packet according to various characteristics of the data packet and/or the data contained in the packet. Then, the network component processes the data packet based on its classification.

The packets of data that flow among the computers that form part of the network can be considered to carry portions of digital information between the different nodes of the network. For example, an application may be running at a computer in one node of the network. The results of such an application may be sent to a computer in a different node of the computer. The information is divided into one or more packets before the data is transferred over the respective network. In the full form of communication, such packets are run upstream and then down stream in order to verify a full response loop. In many systems it is required to analyze the packets flowing back and forth for better network management and system administration, and other application related activities such as billing, prioritizing, and the like.

In a robust high speed network, typically millions of data packets are sent and received by a node ever second. Therefore, the network is required to process millions of such packets. It is clear that these packets must be processed at wire speed so that the transmission of the packet is efficient. For example, in an efficient network the speed of transmission of data will be approximately the same as the speed at which a packet is processed by a node. In other words, the packets are processed at the maximum speed at which they

can be transmitted through the network. The ability to process at wire speed allows the system to work without performance degradation. As the bit rate of the network increases (for example networks capable of speeds 1Gbs are already in use) the number of packets requiring processing also increases.

An important requirement of a good network is that the packets reach their desired destination. The packets should also be prevented from reaching the wrong destination. Theoretically, the best way of doing so is to use the full 104 bit provided as a unique address, however, this approach is impractical as described below.

A large tuple, for example a 104-bit wide tuple, will enable the precise description of the source and destination nodes, the input and output ports as well as the protocol used. However, such a large tuple in turn creates a large address space. For example a 104-bit wide tuple would require an address space of 2^{104} addresses. But then, such a large address space would require addressing a memory using a large tuple. Such a large address would result in huge memory requirements and very inefficient usage of the memory. Therefore, an important challenge is to effectively prevent the need to use such a large tuple for addressing the memory.

A commonly used technique employs hash tables or other hashing techniques. In such a hashing technique, address or location of an identifier is obtained by computing some arithmetic function of the identifier. This arithmetic function is called a hashing function. A hashing function essentially transforms an identifier into an address in the hash table. For example, if X is an identifier, f(X) gives the address of the identifier in the hash table. The memory available to maintain the hash table is normally assumed to be sequential. Also hash

table is normally partitioned into hash buckets. The buckets in turn have one or more slots each capable of holding exactly one record. Therefore, if there are b buckets in the hash table, the transformation f(X) transforms X into an integer 0 through b-1. Since the number of possible identifiers are much larger than the number of buckets there is a distinct possibility that two different identifiers are mapped to the same bucket. For example, if I_1 and I_2 are two different identifiers it is possible that $f(I_1) = f(I_2)$. There is also a likelihood that a bucket can overflow if more identifiers get allotted to the bucket than the number of slots available in the bucket. The desired properties of a hash function are that it be easily computable and that it minimizes the number of collisions.

Several conventional approaches use hashing techniques to store tuples. However, the conventional techniques are deficient in that they lack the capability of truly operating at wire speed. Furthermore, none of the conventional techniques can handle over one million different process flows and the full tuple range. Some of the conventional techniques use several sequential steps that either grow linearly or exponentially with the number of process flows identified. On the other hand, several other conventional techniques require complex resources in order to store tuples using hashing techniques. A disadvantage of the conventional techniques is that they require search mechanisms that are time consuming and impractical for wire speed applications.

Another conventional technique used to store tuples is the use of a Content Addressable Memory (CAM). A CAM is an associative memory device that is capable of searching a list of stored information entries based on the content of the entries. On the other hand, conventional memories search based on the location of the entries in the memory. An information string is provided as

an input to the CAM. The output from the CAM is the address of all of the memory locations in the CAM that contain the particular information string. In conventional memory devices, an address of a memory location is provided as input . The information string contained at that address forms the output. A unique property of CAM memories is their ability to search (i.e., compare to the information string) all of the entries in the CAM table simultaneously. This ability to perform simultaneous searching greatly increases the speed with which memory searches can be performed relative to conventional memories that require a sequential read and compare procedure. The simultaneous search capability is achieved, in part, by including a separate comparator means at each memory location in the CAM device. However, a CAM is complicated in terms of hardware required to implement. For large address spaces a CAM is simply infeasible to implement.

9/5/2003 U.S. Patent No. 5,414,704 ('704) discloses a way of doing source address and destination address lookups for use in a packet data communication system. A way of searching a relatively large database is described, using a combination of programmable hash algorithms. In the hashing technique used in '704 N-bits are hashed into N-bits. Clearly, such a solution is not practical when the address space is large as would be in the case of a communication network. Though '704 discloses the use of a content addressable memory (CAM) in parallel to the hash function, the present invention uses a much improved method.

9/5/2003 U.S. Patent No. 5,708,659 ('659) discloses performing hashing on certain packet headers. In '659 a predetermined number of bits from the packet address information is selected to use a hash key. This hash key is then used to compute a table address. The contents of the table at that address are compared with the packet address information. If it matches, the packet is transmitted

over the port associated with that particular destination address. If it does not match, the table address is incremented by one, and the contents of the new table location identified by the incremented address are compared with the packet address information.

However, '659 but does not teach the method and technique on how to associate a packet with an existing flow in the system. An existing flow in the system refers to one or more packets flowing through the system that was already identified and newly arrived packet needs to be directed to the same packet processor for efficient processing.

9/5/2003 AR

Similarly, in U.S. Patent No. 5,920,900 ('900) a translation is performed by using a programmable hashing technique on an input number to generate a hashed number. In '900, a subset of the hashed number bits are used to index a first hash table. If a collision does not occur in the first hash table, an entry contains an index into an output table which contains the desired translated output number. If a collision occurs, an entry in the first hash table contains a pointer to a first resolution table area in a second hash table. The first resolution table area contains entries which are indexed by additional bits selected from the hashed number in accordance with a mask field in the first hash table location. If collisions occur in the resolution table, a new resolution table is created and the process is repeated. The resolution process thus proceeds in stages until all input numbers have been translated. As can be seen, '900 deals with the problem of packet collision but does not address the issue of association between packets forming a full process flow. Furthermore, all of the above-mentioned conventional techniques posses very limited capabilities of scalability of the solution. Therefore, they are unsuitable for handling the increasing demand for high-speed packet processing at wire speed.

Additionally, it is desirable to associate packet flow with the 7th layer (Application layer) of the Open Systems Interconnection (OSI) 7-layer model. The OSI 7-layer model is a commonly used framework for defining standard for linking heterogeneous computers that form part of a network. OSI uses a concept of layering whereby communication functions are partitioned into a vertical set of layers. Each layer performs a related subset of functions required for communication between two nodes in a computer network system. The seven layers in the model are Physical layer, Data link layer, Network layer, Transport layer, Session layer, Presentation layer and Applications layer. The Physical layer is concerned with the transmission of unstructured bit streams over a physical link. It deals with the mechanical, electrical, and procedural characteristics to establish and maintain the physical link. The Data link layer provides for the reliable transfer of data across the physical link. It sends blocks of data with a necessary synchronization, error control and flow control. The Network layer provides all upper layers with independence from the data transmission and switching technologies used to connect systems. It is responsible for establishing maintaining and terminating connections. The Transport layer avoids reliable and transparent transfer of data between end points. It provides end-to-end error recovery and flow control. The Session layer provides the control structure for communication between applications. It establishes manages and terminates sessions between cooperating applications. The Presentation layer performs useful transformations on data to provide a standardized application interface and provides common communication services. Such services include encryption, text compression and any formatting. Finally the Application layer provides user-level services to the users of the environment. Examples of such services are transaction service, file

transfer, and network management. Therefore, associating packet flow with the 7th layer (Application layer) of the Open Systems Interconnection (OSI) 7-layer model provides for better network management.

II. SUMMARY OF THE INVENTION

To solve the above-mentioned problems in the conventional technologies, it is an object of the present invention is to provide an apparatus which is capable of accepting a packet tuple and uniquely identify it with an existing flow of packets in the system, or alternatively identify it as a new flow. The processing of packet tuples should be performed at wire speeds. By being able to correlate a stream of related packets to a single packet processor, this invention allows the monitoring and management of the system up to the application layer, or the seventh layer of the 7-layer Open Systems Interconnection (OSI) communication model. Furthermore, it allows for the efficient data processing of such packets as all the processing is performed by the same packet processing unit.

It is a further object of this invention to provide a method for transforming a large tuple number associated with a packet into a reduced bit count number while maintaining the unique identification of each flow that the packet is a part of. Another object of this invention is to generate a reduced bit number from a large number based on hashing techniques incorporating certain improvements allowing the generation of "white hashing" numbers. Because of such an improved hashing, though tuples may be highly localized they will still be spread throughout the much more limited range of the hashed numbers. This means that though tuples may poses a locality in the numbers, i.e., are in close

proximity to each other, they will still be spread evenly throughout the limited range of the reduced number range.

To meet the objectives of the present invention there is provided a data packet classifier to classify a plurality of N-bit input tuples, said classifier comprising a hash address generator to generate a plurality of M-bit hash addresses from said plurality of N-bit input tuples, wherein M is significantly smaller than N; a memory having a plurality of memory entries, said memory being addressable by said plurality of M-bit hash addresses, each such address corresponding to a plurality of memory entries, each of said plurality of memory entries capable of storing one of said plurality of N-bit tuples and an associated process flow information; a comparison unit to determine if an incoming N-bit tuple can be matched with a stored N-bit tuple, wherein said associated process flow information is output if a match is found and wherein a new entry is created in the memory for the incoming N-bit tuple if a match is not found.

Preferably the data packet classifier further comprises a content addressable memory (CAM) to store overflowing N-bit tuples and their corresponding process flow information wherein said overflowing N-bit tuple can not be stored in the memory.

Preferably the process flow information in the memory comprises a flow identification number.

Preferably the process flow information in the memory can be updated.

Preferably, an entry in the memory can be deleted.

Preferably searching for an entry in the memory can be ceased when a kill-process command is received.

Preferably the process flow information in the CAM comprises a flow identification number.

Preferably the process flow information in the CAM can be updated.

Preferably an entry in the CAM can be deleted.

Preferably searching for an entry in the CAM can be ceased when a kill-process command is received.

Still preferably, the data packet classifier is further capable of generating a trap if both the memory and the CAM are full.

Still preferably both the memory and CAM are searched in parallel.

Still preferably N > 96.

Still preferably the hash address generator performs hashing on a first 96 bits of an associated N-bit tuple.

Still preferably a comparison of tuple stored in the memory and an incoming tuple is performed using three 32-bit comparators.

Another aspect of the present invention is a network system comprising a plurality of nodes, each of said nodes having a unique N-bit tuple, each of said plurality of nodes comprising a data packet classifier, said address classifier comprising: a hash address generator to generate a plurality of M-bit hash addresses from said plurality of N-bit input tuples, wherein M is significantly smaller than N; a memory having a plurality of memory entries, said memory being addressable by said plurality of M-bit hash addresses, each of said plurality of memory entries capable of storing one of said plurality of N-bit tuples and an associated process flow information; a comparison unit to determine if an incoming N-bit address can be matched with a stored N-bit tuple, wherein said associated process flow information is output if a match is found and wherein a new entry is created in the memory for the incoming N-bit tuple if a match is not found.

Yet another aspect of the present invention is a method of generating an M-bit hash address from an N-bit input tuple comprising: splitting said N-bit input tuple into a first range of X bits and a second range of Y bits wherein X is equal to or smaller than M; applying a hash function to said X bits to generate a white hash address with Z bits wherein Z is equal to or smaller than M; creating said M-bit hash address by combining said Z-bit white hash address and said second range of Y bits using a Boolean operator.

Preferably X is significantly larger than Y.

Preferably X is significantly larger than Z.

Preferably the Boolean operator is an OR.

Preferably the Boolean operator is an XOR.

Preferably the Boolean operation is an AND.

Still preferably N is 104.

Still preferably X is 96 and Y is 8.

Still preferably Z is 20 and M is 20.

Still another aspect of the present invention is a computer program product, including a computer-readable medium comprising instructions, said instructions enabling a computer to perform a hashing function on an N-bit input tuple according to the following steps: splitting said N-bit input tuple into a first range of X bits and a second range of Y bits; applying a hash function to said X bits to generate a white hash address with Z bits; creating said M-bit hash address by combining said Z-bit white hash address and said second range of Y bits using a Boolean operator.

Yet another aspect of the present invention is a computer program product, including a computer-readable medium comprising instructions, said instructions comprising: a hash address generator code to enable a computer

to generate a plurality of M-bit hash addresses from said plurality of N-bit input tuples, wherein M is significantly smaller than N; a memory code to enable a computer to store data in a memory having a plurality of memory entries, said memory code further enabling the computer to address said plurality of M-bit hash addresses, each of said plurality of memory entries capable of storing one of said plurality of N-bit addresses and an associated process flow information; a comparison code to determine if an incoming N-bit tuple can be matched with a stored N-bit tuple, wherein said associated process flow information is output if a match is found and wherein a new entry is created in the memory for the incoming N-bit tuple if a match is not found.

III. BRIEF DESCRIPTION OF THE DRAWINGS

The above and other objects and advantages of the present invention will become more apparent by describing in detail preferred embodiments thereof with reference to the attached drawings in which:

FIG.1 shows a preferred embodiment of a network system according to the present invention.

FIG. 2 is a block diagram of a preferred embodiment of a data packet classifier according to the present invention.

FIG. 3 is a block diagram illustrating the hashing function of the Hash Generator.

FIG.4 shows an example of a tuple or a header.

IV. DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The following description of the preferred embodiments discloses specific configurations, components, and process steps. However, the preferred embodiments are merely examples of the present invention, and thus, the specific features described below are merely used to more easily describe such embodiments and to provide an overall understanding of the present invention. Accordingly, one skilled in the art will readily recognize that the present invention is not limited to the specific embodiments described below. Furthermore, the descriptions of various configurations, components, and steps of the present invention that would have been known to one skilled in the art are omitted for the sake of clarity and brevity.

A. The Network System with a Data Packet Classifier

The preferred embodiment is described using a policy-based network. Such a network system is capable of associating the packets flowing in the system with the Application layer of the 7-layer OSI communication model. It is to be noted that, this ability to associate packet flow with the application it is related to allows for better system and network management. Further, many other enhanced functionality allows for better implementation of a policy-based network. An example of the use of this invention would be in systems that have to provide special priority to packets containing voice over IP or video over IP. Moreover, such a system also provides the ability to route the packets in a defined manner and provide billing specific information. Another example would be differentiated billing system based on the type of application data being transmitted over the network.

An example of a policy-based network system is shown in FIG.1. The network system of FIG.1 comprises hosts 1.1-1.4. A host can be any type of computer including, but not limited to a server 1.2 and 1.3, a workstation 1.4 or a desktop Personal Computer 1.1. The individual hosts are connected using network connections 1.5-1.11. It should be noted that though only four hosts are shown, the network comprises many more hosts. Each host in the network has a unique tuple. Information can ideally flow from any host in the network to any other host in the network.

Each host in the network includes a data packet classifier. The data packet classifier further comprises a hash address generator, a memory and a comparison unit. A preferred embodiment of a data packet classifier is described subsequently with reference to FIG.2. The data packet classifier in such a network system is required to process a substantial number of flows. Additionally, each such flow is associated with a plurality of packets all of which move around the various components of the network.

The host that sends digital information is called the source and the host that receives the information is called the destination. Each message or information flow is divided into one or more packets. Each packet is processed separately and sent from the source to the destination. Such a packet contains a header. The packet header is uniquely identified by a tuple which is 104 bits. Since a bit can have a value of a 0 or 1. Therefore, using 104 bits, 2^{104} tuples possible. However, if each tuple is uniquely identified with its entire number, the number of possible tuples to be handled and classified becomes substantially large for any practical implementation of the system. A real time processing of such a large number of tuples is likely to be impossible.

Extensive simulation studies were performed to determine the maximum possible number of information flows in a network during a short period of time. These simulation studies revealed that that allowing for a million different flows at a given relatively short period of time in a system is sufficient for providing a high quality of service for the system. For more detailed information on these simulations see, for example, Stiliadis, D. "High-Speed Policy-Based Packet Forwarding Using Efficient Multi-Dimensional Range Matching" *ACM SIGCOMM 1998* published in February 1998.

9/5/2003

B. The Data Packet Classifier

FIG.2 shows a preferred embodiment of a data packet classifier. 104 bit tuples are received by a hash address generator 2.4. The hash address generator is capable of transforming the 104 bits of the tuple into a 20 bit long number. Therefore, using the results of the hashing, 2^{20} or more than a million unique digital information flows can be represented. The operation of the Hash Generator is further discussed below. It should be noted that though the preferred embodiment is described using a tuple with 104 bits, tuples of any length can be used without deviating from the spirit of the invention. Likewise, the hash address can be of any length less than the length of the tuple to be within the scope of the present invention.

The 20 bit hash address generated by the Hash Generator is used to access the memory 2.5. Each hash address corresponds to a hash bucket. The memory 2.5 is designed in such a way that each hash bucket is capable of storing eight separate entries. As noted in the background section, it is possible that two different tuples on applying the hashing function result in the same hash

address. This memory is designed so that it has actually eight separate entries for each hush address accessing the memory unit. This specific number of entries for each hash address is chosen based on extensive system simulations balanced with the need to ensure wire speed performance. However, as technology develops, additional entries may be added as necessary. Providing eight different entries for a hash address would ensure eight different entries could be stored in the same hash bucket. That is up to eight different tuples resulting in the same hash address can still be stored in the hash table. This substantially reduces the chance of a tuple being bounced off from the hash table because a particular hash bucket is full.

The hash address generator performs hashing on a tuple associated with an incoming data packet. The result of such a hashing is used to identify a hash bucket that is associated with the incoming tuple. The comparison unit 2.3 which is attached to the CAM, compares data associated with the incoming data packet and the data in each of the eight entries. The comparison unit determines that a successful match has resulted if a match is found between the incoming tuple and an entry in the memory. Upon finding a successful match, the stored process flow information corresponding to the matched entry is output. This process flow information includes a unique ID. If no match is found with an existing entry, a new entry is created corresponding to the incoming packet.

According to another improvement, a CAM is provided as an additional storage facility to deal with incoming packets having tuples that produce a hash address of a hash bucket that is already full. The CAM 2.1 unit is accessed with the full tuple corresponding to the incoming packet. Though remote, it is possible that more than eight unique tuple would hash into a single hash number. The CAM is used for storing process flow information related to such

packets having tuples that produce hash addresses that represent hash buckets that are already full. The CAM is accessed with the full tuple. Simulations have shown that a CAM containing 8K entries is sufficient to reduce the risk of inability to handle a tuple in the classifier unit to a practically negligible number. As systems and technology develops, it is easy to add entries to the CAM to handle additional requirements.

The data packet classifier is also capable of updating the process flow information associated with a data packet. Entries stored in both the memory as well as the CAM can be updated. The data packet classifier is also capable of deleing an entry in the memory.

If the CAM also gets filled up and an incoming packet can not be stored in the memory or the CAM, a trap is generated. Such a trap indicates that the corresponding incoming packet can not be handled by the data packet classifier. Additionally a "kill process" command can be issued to the data packet classifier. On receiving a "kill process" command, the data packet classifier ceases searching the memory as well as the CAM.

As noted before, if in the search of the memory and CAM, which may take place in parallel, no match is found then a new entry is created. Such a new entry is created in the memory if a corresponding hash bucket has not been filled up. If space exists in the memory to store the entry, it is registered in the memory. As part of the entry, full tuple information, the flow identification and other control information is stored. This stored information will be later attached to each incoming packet incoming with a tuple that matches the corresponding entry. After this information is attached, the incoming packet is sent to the packet processor. The ability to update an entry is provided because incoming packets may have undergone further processing. Such a processing

may create a need for making further changes in the stored entry. An update command is used to perform such an updating. When an update command is initiated, the control information associated with the tuple is modified.

In the case where all eight entries of the memory corresponding to a hash bucket that is dedicated for a specific hash address are occupied, the entry is written into the CAM. Simulations have shown that only a fraction of the 8K associated with the CAM is used even when over a million process flow through the system.

An important advantage of the architecture of this system is its ability to easily scale as system demands grow and require handling of millions of more process flow in short periods of time and at wire speed.

C. The Hashing Function

Another aspect of the invention is an improved method to perform hashing that is used by the Hash Address Generator. A preferred embodiment of this hash function is described using Fig.3. Conventionally, the hashing function is applied to all the bits corresponding to a tuple. Instead of applying the hash function on all 104 bits of the tuple, the tuple is divided into two portions, 96 bits of the tuple and the remaining 8 bit indicating the protocol type. Hashing is performed on the 96 bits. Again it should be noted that the fixed number of bits (104, 96, 8, etc) used to describe the preferred embodiment are merely illustrative. The division of bits can be done in any convenient manner without deviating from the spirit of the invention.

While any hash function could be used it is necessary to ensure that the specific hash function used generates a "white hash address". This would mean

that when taking a group of tuples, which in many times can be different in only a minimal way from other tuples, the Hash Generator will still generate hash numbers that are equally spread through the range of address spanned by the 20 bit address space available. While less than the 96 bits could be used for the hash function (in fact, as little as 84 bits would suffice), it is advisable to use the maximum number of bits possible in order to increase the spread and effectiveness of the hash function. The reason for selecting 96 bits specifically is that this allows the use of three 32-bit comparators to compare the incoming tuple and the memory content and get a unique comparison result and further permit the use of six standard 16-bit memory banks to hold the memory content. The hash operation performed on the 96 bits result in a preliminary 20 bit hash address. This is then followed by a logical "OR" or an "XOR" operation performed between the 20-bit result of the preliminary hash operation and the remaining 8 bits from the tuple that did not form part of the hashing operation. Instead of performing a logical "OR" or an "XOR" a Boolean "AND" operation can also be performed. Such a two-step hashing operation results in producing a final hash address that spans the entire 20-bit hash address range.

D. Computer Program Products

Another important aspect of the present invention is a computer program product to enable a computer to perform the hashing operation according to the present invention. The preferred embodiment of such a program product includes a computer-readable medium. The computer-readable medium comprises instructions to enable a computer to perform a hashing function on an N-bit input tuple. The instructions enable the computer to split the N-bit input

tuple into a first range of X bits and a second range of Y bits. The instructions further enable the computer to apply a hash function to the X bits to generate a white hash address with Z bits. Further instructions enable the computer to create a M-bit hash address by combining the Z-bit white hash address and the second range of Y bits using a Boolean operator. The Boolean operator could be a logical "Or" or an "XOR".

Another important aspect of the present invention is a computer program product, including a computer-readable medium comprising instructions to implement the data packet classifier in software. The instructions comprise a hash address generator code to enable a computer to generate a plurality of M-bit hash addresses from a plurality of N-bit input tuples, wherein M is significantly smaller than N. The instructions further comprise a memory code to enable a computer to store data in a memory having a plurality of memory entries. The memory code further enables the computer to address the plurality of M-bit hash addresses. Each of the plurality of memory entries capable of storing one of the plurality of N-bit tuples and an associated process flow information. The instructions further comprise a comparison code to determine if an incoming N-bit tuple can be matched with a stored N-bit tuple. The associated process flow information is output if a match is found and new entry is created in the memory for the incoming N-bit tuple if a match is not found.

It should be noted that the computer readable medium includes any fixed media including, but not limited to, floppy disk, hard disk, CD, chips, tapes, cartridges with ICs, etc. The computer readable media also includes instructions transmitted through a network or downloaded from the Internet.

The previous description of the preferred embodiments is provided to enable a person skilled in the art to make or use the present invention.

Moreover, various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without the use of inventive faculty. Therefore, the present invention is not intended to be limited to the embodiments described herein but is to be accorded the widest scope as defined by the claims and equivalents thereof.